Model Checking LTL with Fairness Assumptions using Net Unfoldings

Prateek Gupta

Department of Computer Sciences
University of Texas at Austin
prateek@cs.utexas.edu

Abstract

We investigate the model checking problem for a class of petri nets using net unfoldings. Net unfoldings are a well studied partial order semantics for petri nets and are well-suited for modeling concurrent and distributed systems. We consider the problem of model checking LTL-X (LTL without the next time operator) with fairness assumptions for the class of one-safe petri nets. Recently, in [1] the authors have proposed a new technique for LTL model checking that exploits the partial order semantics of these net unfoldings. We propose a technique for incorporating fairness assumptions as part of the model while checking LTL properties. Fairness assumptions are often necessary for faithfully modeling a distributed computation. Our method extends the model checking algorithm presented by the authors in [1] and draws machinery from petri net theory and automata theory. Our method has the advantage that it is linear in the number of fairness assumptions and the size of the unfolding developed. Thus, it is practical to check large structures with underlying fairness assumptions.

1 Introduction

Net unfoldings [2], [3] are a well-studied partial order semantics for specifying the behavior of large concurrent and distributed systems. They have the advantage that they can be used to model large state spaces exploiting the partial order semantics of these systems. In a seminal paper [2], McMillan proposed a method for constructing the finite prefix of an unfolding, which maintains complete information about the unfolding. The finite prefix of petri net systems which model large concurrent systems can be exponentially much more succinct as compared to the corresponding Kripke structure (reachability graph) and can be effectively used for

model checking properties of systems. The model checking problem can be formulated as follows: Given a net system Σ and a formula in LTL-X (without the next time operator) specifying the property of interest, we would like to verify if the model satisfies the property. In [1], [4] the authors present an elegant algorithm for model checking LTL-X using net unfoldings. The approach is based on *automata-theoretic approach* to model checking.

The automata theoretic approach to model checking translates the model checking problem into an automata-theoretic problem. Given a structure Σ, the approach translates the structure Σ into a corresponding automaton $\mathcal{A}_\Sigma$ such that the language accepted by the automaton $\mathcal{A}_\Sigma$ is exactly the set of *runs* of the corresponding structure Σ. It is well known that given an LTL formula φ, we can construct a Büchi automaton $\mathcal{A}_{\neg\varphi}$, to accept the set of words which do not satisfy the formula φ [5] . We can then construct a composite automaton $\mathcal{A}_{compose} = \mathcal{A}_\Sigma \times \mathcal{A}_{\neg\varphi}$, which is the synchronized product of the Kripke structure $\mathcal{A}_\Sigma$ and the property automaton $\mathcal{A}_{\neg\varphi}$. The composite automaton, $\mathcal{A}_{compose}$ accepts the intersection of languages of $\mathcal{A}_\Sigma$ and $\mathcal{A}_{\neg\varphi}$, denoted by $\mathcal{L}(\mathcal{A}_\Sigma) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$. Thus, the composite automaton accepts only those *runs* of the original Kripke structure Σ, which do not satisfy the property φ. Thus, the problem of checking if the model satisfies the property reduces to checking the nonemptiness of the composite automaton. However, this approach has the disadvantage that it requires the explicit construction of the Kripke structure which is exponential in the size of Σ. Thus, the approach does not scale well in practise.

In [1], the authors exploit the special structure of the finite prefix of an unfolding to model check a PLTL formula without explicitly constructing the Kripke structure of all the reachable markings of the net system. Instead, given the original net system Σ and the property φ, they construct a composite a net system $\Sigma_{\neg\varphi}$ by synchronizing the original net system and the Büchi automaton for the negation of the property, on the set of visible events of the original net system (We adopt an action based model for model checking LTL formulas). Since the set of visible events is still small compared to the total set of events of the system, the synchronization does not destroy all the concurrency present in the original system. Then, the problem of checking if the model Σ satisfies the property φ, reduces to checking if the language of the composed automaton is empty, where language is defined in some suitable way.

However, the model checking algorithm presented above does not take into consideration fairness constraints which might be imposed on the system to guarantee some sort of liveness properties to the system. Fairness constraints are often necessary for faithfully modeling a distributed system. For example, it is often desired that if a process is enabled *infinitely often* then it is executed infinitely often. The traditional approach for LTL model checking with underlying fairness

assumptions exploits the fact that fairness assumptions are themselves expressible in LTL. Thus, given a property φ and fairness constraints expressed as $fairness$, we instead check for the formula $fairness \Rightarrow \varphi$. However, this algorithm is exponential in the number of fairness constraints and does not scale well in practise for systems with large number of fairness constraints.

We present an alternative technique for incorporating fairness constraints directly as part of the model, rather than the formula. Our method makes use of the automata-theoretic approach to model checking as well as the partial order semantics of net unfoldings. We express fairness assumptions as a subset of the set of visible events, which have to be fairly fired. Our method has the advantage that the size of the Büchi automaton for the property φ is still small and independent of the number of fairness constraints. Our algorithm is linear in the number of fairness constraints and the size of the finite prefix. Thus, it is practical to model check large structures with underlying fairness constraints.

The rest of the paper is organized as follows. In Section 2, we present some definitions related to petri nets and unfoldings. Section 3 describes the syntax of the logic (LTL) and an outline of the automata-theoretic approach for model checking LTL formulas. In Section 4, we present our approach for model checking petri net systems under fairness assumptions. Our approach is based on unfoldings and is a modification of the approach presented in [1]. Finally, we present a summary of our results with future directions of work in Section 6.

2 Petri net definitions

In this section, we briefly review some of the basic definitions related to petri nets. A more detailed description is given in [3].

Petri nets. A net is a triple (P, T, F), where P and T are disjoint sets of places and transitions, respectively, and F is a function $(P \times T) \cup (T \times P) \rightarrow \{0, 1\}$. Places and transitions are generically called *nodes*. If $F(x, y) = 1$ then we say that there is an arc from x to y. The *preset* of a node x, denoted by ${}^\bullet x$, is the set $\{y \in P \cup T | F(y, x) = 1\}$. The *postset* of x, denoted by $x^\bullet$, is the set $\{y \in P \cup T | F(x, y) = 1\}$. In this paper we consider only nets in which every transition has a nonempty preset and a nonempty postset. A *marking* of a net (P, T, F) is a mapping $P \rightarrow N$ (where N denotes the natural numbers including 0). We identify a marking M with the multiset containing $M(p)$ copies of p for every $p \in P$. For instance, if $P = \{p_1, p_2\}$ and $M(p_1) = 1$, $M(p_2) = 2$, we write $M = \{p_1, p_2, p_2\}$.

A marking M enables a transition t if it marks each place $p \in {}^\bullet t$ with a token,

i.e. if $M(p) > 0$ for each $p \in {}^\bullet t$. If t is enabled at M, then it can fire or occur, and its $occurrence$ leads to a new marking M' , obtained by removing a token from each place in the preset of t, and adding a token to each place in its postset; formally, $M'(p) = M(p) - F(p,t) + F(t,p)$ for every place p. For each transition t the relation $-\!\!\rightarrow^t$ is defined as follows: $M \longrightarrow^t M'$ if t is enabled at M and its occurrence leads to M'.

A 4-tuple $\Sigma = (P, T, F, M_0)$ is a *net system* if (P, T, F) is a net and M_0 is a marking of (P, T, F) (called the *initial marking* of Σ). A sequence of transitions $\sigma = t_1 t_2 \ldots t_n$ is an *occurrence sequence* if there exist markings $M_1, M_2, \ldots, M_n$ such that

$$M_0 \longrightarrow^{t_1} M_1 \longrightarrow^{t_2} \ldots M_{n-1} \longrightarrow^{t_n} M_n$$

M_n is the marking reached by the occurrence of σ, which is also denoted by $M_0 \longrightarrow^\sigma M_n$. A marking M is a *reachable marking* if there exists an occurrence sequence σ such that $M_0 \longrightarrow^\sigma M$. An *execution* is an infinite occurrence sequence starting from the initial marking. The *reachability graph* of a net system Σ is the labeled graph having the reachable markings of Σ as nodes, and the $-\!\!\rightarrow^t$ relations (more precisely, their restriction to the set of reachable markings) as edges. For the purpose of analysis, we only consider systems with finite reachability graphs.

A marking M of a net is *n-safe* if $M(p) \leq n$ for every place p. A net system is *n-safe* if all its reachable markings are n-safe. For the purpose of model checking LTL properties we restrict our attention to one-safe nets. Figure 3 shows an example of a one-safe net system.

Occurrence nets. Let (S, T, F) be a net and let x and y be two nodes of the net. We say that x is causally related to y, denoted by $x \leq y$, if there is a path from x to y. We say that x and y are in conflict, denoted by $x \# y$, if there is a place z, different from x and y, such that there are two distinct paths to x and y from z, which immediately branch at z. Finally, we say that x and y are concurrent, denoted by $x \; co \; y$, if neither $x \leq y$ nor $y \leq x$ nor $x \# y$ hold. A $co\text{-}set$ is a set of nodes X such that $x \; co \; y$ for every $x, y \in X$.

An *occurrence net* is a net $N = (B, E, F)$ satisfying the following three conditions:

- for every $b \in B$, $|{}^\bullet b| \leq 1$,
- F is acyclic, $i.e.$, the irreflexive transitive closure of F is a partial order.

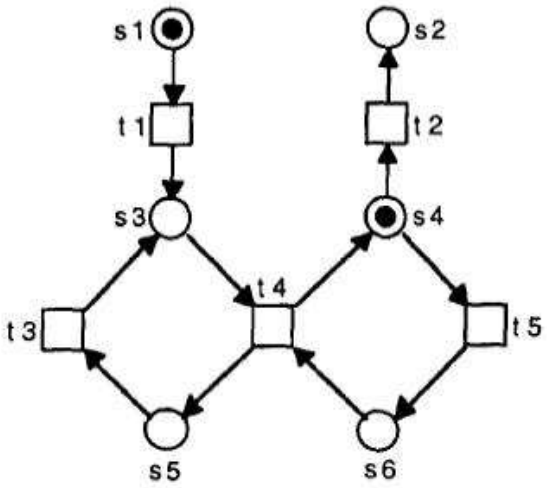

Figure 1: A one-safe net system

- N is finitely preceded, *i.e.*, for every $x \in B \cup E$, the set of elements $y \in B \cup E$ such that (y, x) belongs to the (irreflexive) transitive closure of F is finite, and
- no event $e \in E$ is in self-conflict.

A place of an occurrence net is *minimal* if it has no input transitions. The *default initial marking* of an occurrence net puts one token on each minimal place and none in the rest.

Branching processes. Branching processes are the "unfoldings" of a net system. Given a net system $\Sigma = (S, T, F)$, we associate with Σ a labeled occurrence net $N = (B, E, F)$, called a *branching process* of Σ. The conditions and events of N are labeled with places and transitions of Σ, respectively. We define a *homomorphism* from Σ to N as a mapping $h : S \cup T \to B \cup E$ such that:

- $h(B) \subseteq S$ and $h(E) \subseteq T$, and
- for every event $e \in E$, the restriction of h to ${}^\bullet t$ is a bijection from ${}^\bullet t$ and ${}^\bullet h(t)$, and similarly for $t^\bullet$ and $h(t)^\bullet$.

Informally, h is a mapping that preserves the nature of places and environment of transitions. We represent a condition as a pair (p, e), where p is the label of

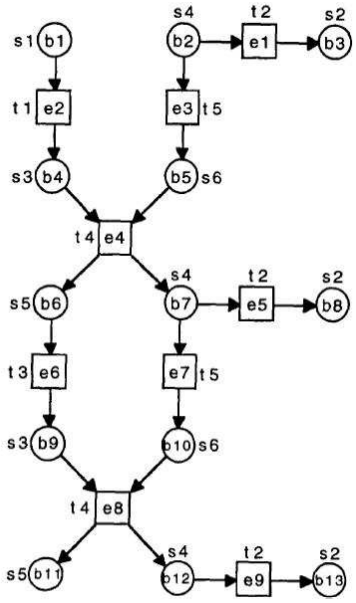

Figure 2: An unfolding of the net system shown in Figure 1

the condition and e is the corresponding input event. We use the special symbol $\perp$ to denote a condition $(p, \perp)$ which has no input event. Similarly, for an event $e = (t, X)$, t is the label of the event and X is the corresponding co-set of input conditions.

The set of finite branching processes of a net system Σ with the initial marking $M_0 = \{s_0, s_1, \ldots, s_n\}$ is defined inductively as follows:

- $(\{(s_1, \perp), \ldots, (s_n, \perp)\}, \phi)$ is a branching process of Σ,
- If (B, E) is a branching process of Σ, $t \in T$, and $X \subseteq B$ is a co-set labeled by ${}^\bullet t$, then $(B \cup \{(p, e) | p \in t^\bullet\}, E \cup \{e\})$ is also a branching process of Σ, where $e = (t, X)$. If $e \notin E$, then e is called a *possible extension* of (B, E).

Remark. There exists a unique *maximal* branching process up to isomorphism, called the *unfolding* of Σ.

Configurations and Cuts. A *configuration* C of an occurrence net $N = (B, E, F)$ is a set of events satisfying the following two conditions:

- C is causally closed,*i.e.*, if $e \in C$ then $\forall e' \leq e, e' \in C$, and

- C is conflict free, $i.e.$, $\nexists e, e' \in C$, such that $e \# e'$.

Given an event e, we call $[e] = \{e' \in E | e' \leq e\}$ the *local configuration* of e. We denote by $Min(N)$, the set of minimal places of N. Given a branching process $\beta = (N, p)$ of a net system Σ, a configuration C of the branching process is associated with a set of conditions called a cut, defined as follows:

$$Cut(C) = ((Min(\beta) \cup C^{\bullet}) \backslash {}^{\bullet}C).$$

The marking associated with the configuration C, denoted by $Mark(C)$ is the set $l(Cut(C))$, where l is the labeling function which labels each condition and event of β, with the corresponding places and transitions of Σ, respectively.

Given a configuration C, we define $\Uparrow C$ as the pair (N', p'), where N' is the unique subset of N whose set of nodes is $\{x | (\exists y \in C : x \geq y) \wedge \forall y \in C : \neg(x \# y)\}$ and p' is the restriction of p to the nodes of N'. Intuitively, $\Uparrow C$ denotes the set of branching processes which can be obtained with $Mark(C)$ as the initial marking of the net system Σ. Given a configuration C, we denote by $C \oplus E$ the set $C \cup E$, under the condition that $C \cup E$ is a configuration and $C \cap E = \phi$. We say that $C \oplus E$ is an *extension* of C. Let C_1 and C_2 be two configurations and let $Mark(C_1) = Mark(C_2)$, then we can define an isomorphism $I_{C_1}^{C_2}$ from $\Uparrow C_1$ to $\Uparrow C_2$. Intuitively, if two configurations C_1 and C_2 lead to the same marking then the branching processes, $\Uparrow C_1$ and $\Uparrow C_2$ are basically the same up to isomorphism. Thus, for two finite configurations C_1,C_2 having $Mark(C_1) = Mark(C_2)$, we have an isomorphism from the finite extensions of C_1 to the finite extensions of C_2.

Adequate orders. A partial order $\prec$ on the finite configurations of the unfolding of a net system Σ is an *adequate order* if:

- $\prec$ is well-founded,
- $\prec$ is a refinement of $\subseteq$,i.e., $C_1 \subset C_2$ implies $C_1 \prec C_2$, and
- $\prec$ is preserved by finite extensions, i.e., if $C_1 \prec C_2$ and $Mark(C_1) = Mark(C_2)$, then the isomorphism $I_{C_1}^{C_2}$ from above satisfies $C_1 \oplus E \prec C_1 \oplus I_{C_1}^{C_2}(E)$ for all finite extensions $C_1 \oplus E$ of C_1.

Complete finite prefix. A complete finite prefix $\mathcal{F} = (B, E, F)$ of a net system $\Sigma = (S, T, F)$ is a finite part of the unfolding of Σ which satisfies the following properties:

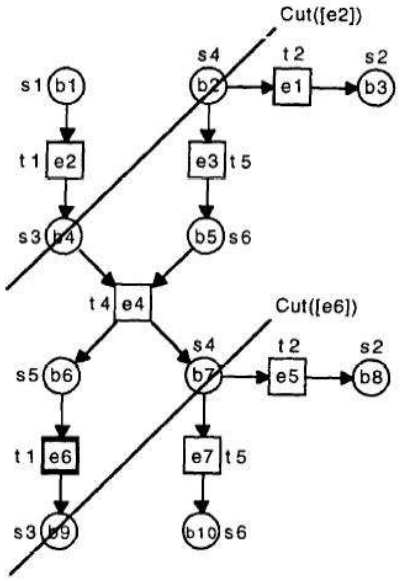

Figure 3: Configurations and Cuts of the one-safe net system shown in Figure 1

- Every reachable marking of the original net system Σ is a reachable marking of $\mathcal{F}$.
- For every transition $t \in T$, which is enabled in some marking M of Σ, there is an event $e \in E$ labeled with t, such that e is enabled in marking M of $\mathcal{F}$.

We say that the finite prefix $\mathcal{F}$ satisfies the completeness property, described above. Thus, a finite prefix has complete information about the net system and has the advantage of being exponentially much more succinct than the corresponding Kripke structure (reachability graph) of the net system. Although, in the worst case the size of the prefix can be as large as the number of reachable markings of a one-safe net system, as described in [3]. For the purpose of analysis, we shall restrict our attention to complete finite prefixes of one safe net systems.

3 Automata theoretic approach to LTL model checking

Linear temporal logic. Linear temporal logic can often be used to describe safety properties of a system. For a given system, the formulas of linear temporal logic are interpreted over infinite runs of the system. An infinite run ρ of the system

is defined as an infinite sequence of states $\rho = s_0, s_1, s_2, \ldots$ where each state s_i assigns an interpretation to the variables mentioned in the formula.

Given a finite nonempty set of propositions AP, LTL formulas are defined inductively as follows:

Every member $p \in AP$, is a formula.

If φ and ψ are LTL formulas, then $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \mathcal{U} \psi$ are LTL formulas.

We introduce the following abbreviations defined in the usual way: $\Diamond\varphi = true\mathcal{U}\varphi$ (possibly φ) and $\Box\varphi = \neg\Diamond\neg\varphi$ (always φ). We also assume the boolean abbreviations $\vee$, $\Rightarrow$ and $\Leftrightarrow$ are defined in the usual manner.

An LTL formula is interpreted over an infinite word $\rho = x_o x_1 x_2 \ldots$ over the alphabet 2^{AP}, i.e. a mapping $\rho : N \rightarrow 2^{AP}$. The mapping is interpreted in the usual way with the propositions labeling a state as $true$, and the rest of the propositions are interpreted as $false$. We denote by ρ_j, the suffix $x_j x_{j+1} \ldots$ holding at all times $t \geq j$. We denote by $\rho \models \varphi$, the fact that the run ρ satisfies the formula φ. Then,

$\rho \models p$, if $p \in x_0$ is an atomic proposition, $p \in AP$,
$\rho \models \neg\varphi$, if it is not the case that $\rho \models \varphi$,
$\rho \models \varphi \wedge \psi$, if $\rho \models \varphi$ and $\rho \models \psi$,
$\rho \models \varphi \mathcal{U} \psi$, if there exists some index $j \geq 0$, such that $\rho_j \models \psi$ and for all $i \leq j$, we have $\rho_i \models \varphi$,
$\rho \models \Diamond\varphi$, if for some $j \geq 0$ $\rho_j \models \varphi$,
$\rho \models \Box\varphi$, if for all $i \geq 0$, $\rho_i \models \varphi$.

Given a net system $\Sigma = (S, T, F)$, we assign *labels* to the set of transitions of the net system, with the help of a labeling function L. Given a labeled net system (S, T, F, L), we identify a set $\mathcal{V} \subseteq L$ of the set of labels L. Transitions which are labeled with the set of labels from $\mathcal{V}$ are said to be *visible* (observable) transitions and the rest are called *hidden* transitions. We replace the labels of the hidden transitions with a special value τ. Given an infinite occurrence sequence $\sigma = t_1, t_2, \ldots$, we define the ω-word $\omega(\sigma) = L(t_1), L(t_2), \ldots$. Given two ω-words ω_1 and ω_2, they are said to be *stuttering equivalent* if one can be obtained from the other by stuttering τ-transitions only. Given a net system Σ, and a set of visible transition labels $\mathcal{V}$, the set of LTL formulas over Σ are defined inductively as above with the set of atomic propositions AP equal to $\mathcal{V}$. The formulas of LTL over Σ with the initial marking M_0 are interpreted over the set of ω-words (up to stuttering) defined by all possible occurrence sequences of the net system Σ. Formally, an execution $M_0 \longrightarrow^{t_1} M_1 \longrightarrow^{t_2} \ldots$, of Σ satisfies a formula φ if the corresponding ω-word $L(t_1), L(t_2), \ldots$ satisfies the formula φ. The net system Σ satisfies φ, denoted by $\Sigma \models \varphi$, if every execution of Σ satisfies φ. Without loss of generality, we shall assume that the flow function F is total, *i.e.*, all executions of Σ are infinite.

Automata-theoretic approach to model checking. Given a model (Kripke structure) M and an LTL formula φ, the traditional automata-theoretic approach to model checking, consists of the following three steps:

- Given the formula φ, we first construct a Büchi automaton $\mathcal{A}_{\neg\varphi}$ to accept the negation of φ.
- Construct a composite automaton $\mathcal{A}_{composite} = M \times \mathcal{A}_{\neg\varphi}$, which is defined as the synchronous product of the model and the Büchi automaton $\mathcal{A}_{\neg\varphi}$.
- The problem of deciding if $M \models \varphi$, is reduced to checking the nonemptiness of language accepted by the composite automaton. We denote the language accepted by the composite automaton as $\mathcal{L}(A_{composite})$. Now, $M \models \varphi$ iff $\mathcal{L}(A_{composite}) = \phi$

However, the traditional approach requires the explicit construction of the Kripke structure. For a given one-safe net system Σ, the Kripke structure is the reachability graph and might be exponential in the size of Σ. Thus, this approach is not directly suitable for application to petri nets. In [1], the authors present an unfolding based approach for model checking a one-safe net system. Instead of explicitly constructing the reachable state space of the net system, a composite net system is constructed by synchronizing the original net system with the Büchi automaton for the property φ. Then the nonemptiness of the language of the composite net system is tested by constructing complete finite prefixes of the net system. The finite prefixes so constructed are then checked to satisfy some special conditions. The original system Σ satisfies the property φ iff these special conditions are met. We describe in some detail the approach presented in [1]. The modified procedure consists of the following three steps:

- We first construct a Büchi automaton $\mathcal{A}_{\neg\varphi}$ to accept the negation of φ.
- We then construct a composite net system $\Sigma_{\neg\varphi}$ by composing the net Σ with $\mathcal{A}_{\neg\varphi}$. The composition is defined as follows: For every state s of the Büchi automaton $\mathcal{A}_{\neg\varphi}$, we add a place s to Σ. For every transition t of Σ, which is labeled by $l \in \mathcal{V}$, if there is a corresponding transition in the Büchi automaton of the form $s \longrightarrow^{l} s'$, then we add additional arcs so that $s \in {}^\bullet t$ and $s' \in t^\bullet$. Thus, we synchronize the net with the Büchi automaton on the set of *visible* transitions.
- Finally, we check the emptiness of the language accepted by $\Sigma_{\neg\varphi}$,denoted by $\mathcal{L}(\Sigma_{\neg\varphi})$, where language is defined in some suitable way. The original net system satisfies φ, denoted by $\Sigma \models \varphi$ iff $\mathcal{L}(\Sigma_{\neg\varphi}) = \phi$.

In what follows, we shall refer to the places and transitions of $\Sigma_{\neg\varphi}$ corresponding to the Büchi automaton as *Büchi places* and *Büchi transitions*. From the above construction, it is easy to see that $\Sigma \not\models \varphi$ if there exists an occurrence sequence of $\Sigma_{\neg\varphi}$, such that a final state of the Büchi automaton is visited infinitely often. Let I denote the set of places of $\Sigma_{\neg\varphi}$ which correspond to final states of the Büchi automaton. We have the following definitions.

Definition 1. *Infinite Trace Monitors and Livelock Monitors*

- The set I_T of *infinite trace monitors* is the set of transitions $\{t | t^\bullet \cap I \neq \phi \ \wedge L(t) \in \mathcal{V}\}$, *i.e.*, I_T denotes the set of visible transitions which visit a final state of the Büchi automaton.
- The set I_L of *livelock monitors* is defined as the set of transitions t such that there exist states s, s' of the Büchi automaton $\mathcal{A}_{\neg\varphi}$ such that $s \in {}^\bullet t, s' \in t^\bullet$ and $\mathcal{A}_{\neg\varphi}$ accepts an infinite sequence of invisible transitions from s' and $L(t) \in \mathcal{V}$.

We now define the notion of illegal ω-traces and illegal livelocks.

Definition 2. *Illegal ω-Traces and Illegal Livelocks*

- An *illegal ω-trace* of $\Sigma_{\neg\varphi}$ is an infinite occurrence sequence $\sigma = t_0, t_1, \ldots$ such that $t_i \in I_T$ for infinitely many indices i.
- An *illegal livelock* of $\Sigma_{\neg\varphi}$ is an infinite occurrence sequence $\sigma = t_0, t_1, \ldots, t_i, t_{i+1}, \ldots$ such that $t_i \in I_L$, and $t_{i+k} \in T \backslash \mathcal{V}$.

The main result of [1] is Theorem 1.

Theorem 1. *Let Σ be a labeled net system, and φ be a LTL formula. We say that Σ satisfies the formula φ, denoted by $\Sigma \models \varphi$, iff $\Sigma_{\neg\varphi}$ has no illegal ω-traces and no illegal livelocks.*

Intuitively, an illegal ω-trace corresponds to a run of the Büchi automaton $\mathcal{A}_{\neg\varphi}$ in which some final state is visited infinitely often. An illegal livelock, on the other hand, corresponds to a run in which there are only finitely many visible transitions. Thus, any infinite run of $\Sigma_{\neg\varphi}$ is accepting iff there are infinitely many visible transitions in the run and no final state of $\mathcal{A}_{\neg\varphi}$ is visited infinitely often along the run.

4 LTL model checking under fairness assumptions

Fairness assumptions are often necessary for faithfully modeling distributed systems with the usual interleaving semantics. In semantics of Petri nets, a branching process can represent an individual process while the unfolding represents a set of branching processes, namely, a distributed computation. It is usually desired that if a process is enabled *infinitely often*, it is executed *infinitely often*. The above condition is called as *strong fairness*. In the remainder of this section, we shall formally define the various forms of fairness assumptions and present an alternative approach to handle them. Since, we follow the action based model, we shall restrict our attention to fair transitions. Many forms of fair transition systems have been proposed in the literature. A fair transition system includes a set of compassionate transitions also called *strongly fair transitions* and a set of just transitions also called *weakly fair transitions*. A strongly fair transition requirement for a transition t means that if t is enabled infinitely often, it is executed infinitely often. On the other hand, a weakly fair transition imposes the restriction that if t is enabled continuously, it is eventually executed. In temporal logic notation, fairness requirements are path properties expressible as follows:

- $\Box(\Diamond\Box e\text{-}enabled \Rightarrow \Diamond e\text{-}executed)$, for weak fairness meaning that *if event e is continuously enabled, it will be eventually executed.*
- $\Box(\Box\Diamond e\text{-}enabled \Rightarrow \Diamond e\text{-}executed)$, for strong fairness meaning that *if event e is infinitely often enabled, it will be eventually executed.*[1]

In the past, various methods have been presented to cope with fairness assumptions when model checking LTL under fairness assumptions. The traditional approach for handling fairness assumptions exploits the property that fairness assumptions are expressible in LTL. Thus, if fairness assumptions are expressible as $fairness$ and the desired property is expressible as $property$ in LTL, the model is verified by checking the formula $fairness \Rightarrow property$. However, this method has the disadvantage that the size of the Büchi automaton for the formula can be exponential in the size of the formula. An alternative approach for model checking B event systems under fairness assumptions is presented in [6]. This approach directly checks the model for the property by incorporating the fairness assumptions as part of the model rather than the formula to be checked. However, there exists no previous work, to the best of our knowledge, which applies this approach to petri nets and unfoldings. We present an approach for directly checking LTL properties under fairness assumptions using net unfoldings, without including its

[1] Notice that we adopt a slightly different notion of strong fairness, but, our definition of strong fairness is more restrictive and implies the usual definition of a strongly fair transition.

specification in the LTL formula. In what follows, we shall use events and transitions interchangibly.

Let $\Sigma = (P, T, F, l, M_0)$ be a labeled petri net system, where P and T denote the set of places and transitions, $F : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ is the flow function, $l : P \cup T \rightarrow \mathcal{L}$ is the labeling function, and M_0 is the initial marking. Let us denote by $\Sigma_f = (\Sigma, H)$, where $H \subseteq \mathcal{L}$ is a set of strongly fair transitions, the fair petri net system consisting of a set of additional strongly fair transitions. Here, we do not distinguish between strong fairness and weak fairness, since weak fairness is a particular case of strong fairness. Before we introduce our algorithm, we have the following definition.

Definition 3. *A cycle $M_i \longrightarrow^{e_i} M_{i+1} \longrightarrow^{e_{i+1}} \ldots \longrightarrow^{e_n} M_n$ (in the reachability graph of Σ), where $M_n = M_i$, is a* fair exiting cycle *with respect to a fairness hypothesis h, if there there exists a transition t ($t \in T_h$) s.t. $l(t) = h$ and further t is a cycle exiting transition, and for all $j \in [i, n]$, $l(e_i) \neq h$.*

In words, a fair exiting cycle for a fairness hypothesis h is the one which does not contain any h labeled transition and further it has some cycle exiting transition labeled as h. Intuitively, a fair exiting cycle violates the fairness assumption, thus, it trivially satisfies the LTL formula $fairness \Rightarrow property$.

The set T_H of transitions, corresponding to the set H, is the set of events $\{e | l(e) \in H\}$. We call an individual member of H, a fairness hypothesis. The set T_h of fair transitions for a given fairness hypothesis $h \in H$, is the set of events e such that $l(e) = h$ and further e is a *cycle exiting transition*

We have the following result:

Theorem 2. *Let $\Sigma_f = (\Sigma, H)$ be a fair net system ,where Σ is a one safe labeled net system and H is a set of strong fairness assumptions. Let φ be a LTL formula expressing the property. We say that Σ_f satisfies φ under the set of fairness assumptions, denoted by $\Sigma_f \models \varphi$, if and only if the following two conditions are satisfied:*

- *Σ does not have a non-fair exiting accepting cycle, i.e., Σ does not have an illegal ω-trace which satisfies the fairness constraints.*
- *Σ has no illegal livelocks.*

Proof. The outline of the proof is shown below.

(*If direction*). We say that Σ satisfies φ (without the fairness constraints), denoted by $\Sigma \models \varphi$, if it has no illegal ω-traces and no illegal livelocks (this follows directly from Theorem 1). We now restrict our attention to fair executions of Σ.

We say that an illegal ω-trace ρ is fair provided it executes some Büchi transition infinitely often and additionally satisfies the fairness constraint, ı.e., for a given fairness hypothesis h, if some transition labeled h is enabled infinitely often during the execution of ρ then some transition labeled h is executed infinitely often in ρ. It easily follows from the definition of a fair illegal ω-trace and the proof of Theorem 1, that the fair net system Σ_f satisfies φ, denoted by $\Sigma_f \models \varphi$, if it has no fair illegal ω-traces and illegal livelocks.

(*Only-if direction*). From Theorem 1 it follows that if Σ has no illegal livelocks and illegal ω-traces, then Σ (not Σ_f) satisfies φ . We note that non-fair ω-traces do not satisfy the fairness constraints. Thus, they are trivially accepted since we only restrict our attention to fair executions of Σ. Moreover, the set of illegal livelocks and illegal ω-traces is disjoint (by definition). Thus, it follows that if Σ has no fair illegal ω-trace and no illegal livelocks (we are only allowing non-fair ω-traces), then the fair net system Σ_f satisfies φ.

Thus, we only restrict our attention to fair illegal ω-traces and illegal livelocks of Σ. In the next section we give a procedure for generating a tableau which can be efficiently used for detecting fair illegal ω-traces and illegal livelocks.

5 Tableau system

We showed in Section 4 that the model checking problem for the fair net system $\Sigma_f = (\Sigma, H)$, with H as the set of fairness assumptions, can be reduced to the detection of fair illegal ω-traces and illegal livelocks. We also note that both these conditions are mutually exclusive by definition, thus the existence of one is not affected by the other. The tableau generation procedure outlined in this section is similar to the one outlined in [1]. We have modified the procedure for detecting illegal ω-traces to identify fair ω-traces only. The procedure for detecting illegal livelocks remains essentially unchanged, since both these conditions are mutually exclusive.

5.1 Tableau system for fair illegal ω-trace problem

We present an algorithm for the detection of fair illegal ω-traces using the notion of total adequate orders for one-safe petri nets. Total adequate orders for one-safe nets have been proposed in [1] [4]. Our algorithm takes as a parameter the total adequate order $\prec$ and outputs the tableau constructed. A tableau is essentially a complete finite prefix which satisfies some additional constraints.

Definition 4. *An event e of a branching process BP is called a repeat (with respect to $\prec$) if BP contains another process e', called the image of e, such that $Mark([e]) = Mark([e'])$ and one of the following two exclusive conditions are met:*

- *(Type I) $e' < e$*
- *(Type II) $\neg(e' < e)$, $[e'] \prec [e]$, and $\#_I[e'] \geq \#_I[e]$. ($\#_I(C)$ for a configuration C denotes the no. of* Büchi events *in C).*

*A terminal is a minimal repeat with respect to the causal relation,*i.e, *a repeat e is a terminal if the unfolding of Σ contains no other event e' such that $e' < e$. A repeat e with image e' is said to be successful if it is of type I (*i.e. *it satisfies condition I) and additionally $[e]\backslash[e']$ forms a non-fair exiting cycle.*

A tableau is a branching process BP such that for every possible extension of BP one of the immediate causal predecessors is a terminal. The tableau is called successful if it contains a successful terminal.

The tableau generation procedure is similar to the one presented in [1]. However, we have modified the procedure to detect fair ω-traces. Intuitively, a tableau represents a complete finite prefix in which the terminals represent the cut-off events. In the case of terminals of type I, we have $e' < e$ and further $[e]\backslash[e']$ forms a non-fair exiting accepting cycle. Thus, we have a finite witness of a fair illegal ω-trace since the configuration $[e]\backslash[e']$ can now be pumped infinitely often to get an infinite trace which is fair and visits a final state of the Büchi automaton infinitely often. In other words, since $Mark([e]) = Mark([e'])$ and e' happened before e and the configuration $[e]\backslash[e']$ contains an I-event and also satisfies the fairness constraints, we can now pump the configuration $[e]\backslash[e']$ infinitely often to get a fair ω-trace. The second condition is required for completeness and we refer the reader to [1] for a more detailed discussion.

Theorem 4. *Let ξ be a tableau constructed for a net system Σ according to the total adequate order $\prec$, then Σ contains a fair illegal ω-trace iff ξ is successful.*

Proof. The soundness of the theorem is straightforward. Let ξ be a tableau constructed by the above procedure. Now if ξ is successful then it contains a successful terminal (of type I). Let e be a successful terminal of type I and let e' be the corresponding image of e. By condition I, we have $e' < e$ and $[e]\backslash[e']$ forms a non-fair exiting accepting cycle, *i.e.*, the configuration $[e]\backslash[e']$ contains an I-event and additionally satisfies the fairness constraints. Since the markings of $[e']$ and $[e]$ coincide, we now pump the configuration $[e]\backslash[e']$, between markings $Mark([e'])$ and $Mark([e])$, infinitely often. Thus, we have a fair illegal ω-trace. The proof

of completeness follows directly from [1] since conditions I and II are mutually exclusive. Intuitively, condition II ensures that if C is a configuration which satisfies condition II and is a witness of an illegal ω-trace then there exists another configuration $C' < C$ which is also a witness of the illegal ω-trace. The same holds for a fair illegal ω-trace. Since the set of configurations is well-founded we will eventually terminate in a configuration satisfying condition I.

5.2 Tableau system for illegal livelock problem

In this section, we provide a description of the tableau construction procedure for the illegal livelock problem. The procedure is essentially the same as presented in [1] (since the detection of illegal ω-traces and illegal livelocks are disjoint by definition). However, we review some of the basic definitions and outline the main procedure for completeness sake.

Definition 5. *A marking M belongs to the set CP of checkpoints of Σ if $M = Mark([e])$ for some non-terminal event e of the complete prefix of Σ labeled by a transition of I_L.*

Let $\{M_1, M_2, \ldots, M_n\}$ be the set of checkpoints obtained above. Let Σ_i denote the net system Σ after removing all visible transitions together with incident arcs, with the initial marking M_i.

Definition 6. *Let $BP_1, \ldots, BP_n$ be branching processes of $\Sigma_1, \ldots, \Sigma_n$, respectively. An event e of BP_i is a repeat (with respect to $\prec$), if there is an event e' in BP_j ($j \leq i$), called the image of e, such that $Mark([e]) = Mark([e'])$ and one of the following three exclusive conditions are met:*

- *(Type I) $j < i$,*
- *(Type II) $i = j$ and $e' < e$,*
- *(Type III) $i = j$, $\neg(e' < e)$,$[e'] \prec [e]$, and $|[e']| \geq |[e]|$.*

A repeat e of BP_i is a terminal if BP_i contains no repeat $e < e'$. A repeat e with image e' is successful if it is of type II.

A tableau is a tuple $BP_1, \ldots, BP_i$ of branching processes of $\Sigma_1, \ldots, \Sigma_n$ such that for every $1 \leq i \leq n$ and every possible extension of BP_i one of the immediate causal predecessors is a terminal. The tableau is called successful if it contains a successful terminal.

We state the following result, without proof:

Theorem 5. *Let $\xi = (BP_1, \ldots, BP_n)$ be a tableau constructed for a net system Σ according to the total adequate order $\prec$, then Σ contains an illegal livelock iff ξ is successful.*

6 Conclusions and future work

We presented an approach for model checking a petri net system under a set of fairness assumptions. The model checking problem (for linear temporal logic) for petri nets is PSAPCE-complete in general [7]. However, our approach is based on unfoldings of a net system, which can be exponentially much more succinct as compared to the corresponding Kripke structure (reachability graph), in general. Thus, our approach has the advantage that although the Kripke structure may be very large, the unfoldings are typically much smaller. We adopt an action based model for model checking LTL formulas. Our approach is a modification of the approach presented in [1]. We present a technique for incorporating the fairness assumptions as part of the model, rather than the formula. Fairness assumptions are often necessary for faithfully modeling a distributed computation. The traditional automata-theoretic approach exploits the fact that fairness assumptions are expressible in LTL and checks for $fairness \Rightarrow property$. We present an alternative approach for model checking under fairness assumptions without including its specification in the formula. Our method has the advantage that it is linear in the number of fairness assumptions and the size of the unfolding. Thus, it is practical to check large structures with underlying fairness assumptions.

References

[1] Javier Esparza and Keijo Heljanko. A New Unfolding Approach to LTL Model Checking. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 475–486. Springer-Verlag, 2000.

[2] Kenneth L. McMillan. A Technique of State Space Search Based on Unfolding. *Formal methods in System Design*, 6(1):45–65, 1995.

[3] Javier Esparza, Stefan Romer, and Walter Vogler. An Improvement of McMillan's Unfolding Algorithm. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 87–106. Springer-Verlag, 1996.

[4] Javier Esparza and Keijo Heljanko. Implementing LTL Model Checking with Net Unfoldings. Research Report A68, Helsinki University of Technology, 2001.

[5] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly Automatic Verification of Linear Temporal Logic. In *Proceedings of 15th Workshop Protocol Specification, Testing, and Verification*, pages 3–18, 1995.

[6] Françoise Bellegarde, Samir Chouali, and Jacques Julliand. Verification of Dynamic Constraints for B Event Systems under Fairness Assumptions. In *ZB*, pages 477–496, 2002.

[7] Keijo Heljanko. Model Checking with Finite Complete Prefixes Is PSPACE-Complete. In *Proceedings of the International Conference on Concurrency Theory*, 2000.